

This và Function nâng cao

This

this trong Javascript là từ khóa đề cập đến object mà nó thuộc về.

this trong một phương thức (method)

Trong phương thức, **this** đề cập đến **object chủ quản**

```
var person = {
  firstName: 'John',
  lastName: 'Doe',
  id: 5566,
  fullName: function () {
    return this.firstName + ' ' + this.lastName
  }
}
var handleFullName = person.fullName
console.log(person.fullName()) // John Doe
console.log(handleFullName()) // undefined undefined
```

this đứng một mình

Khi đứng một mình, **this** đề cập đến **global object**. Nếu là trình duyệt thì sẽ là **[object Window]**

```
var x = this
console.log(this) //[object Window]
```

this ở trong một function

Nếu mặc định thì **this** sẽ đề cập đến **global object**

```
function myFunction() {
  return this
}
console.log(myFunction()) // [object Window]
```

Nếu trong chế độ **'strict mode'** thì sẽ là **undefined**

```
'use strict'
function myFunction() {
  return this
}
```

```
}  
console.log(myFunction()) // undefined
```

Để rõ ràng thì chúng ta sẽ ví dụ trong 'strict mode' thôi nhé Trong một constructor function cũng tương tự như vậy.

```
'use strict'  
function Car(name) {  
  this.name = name  
  this.printName = function () {  
    console.log(this.name)  
  }  
}  
Car('bmw') // Lỗi vì không thể truy cập thuộc tính name của undefined  
const bmw = new Car('BMW')  
bmw.printName() // log ra BMW
```

this ở trong một Event Handler

Trong một HTML event handler, **this** đề cập đến HTML element mà nó nhận event.

Khi nhấn vào button dưới đây thì nó sẽ được set **display:none**

```
<button onclick="this.style.display='none'">Click to Remove Me!</button>
```

this ở trong callback

this trong đoạn code này sẽ không đề cập đến object **delay**

```
const delay = {  
  lastName: 'Duoc',  
  print() {  
    setTimeout(function () {  
      console.log(this.lastName) // undefined  
    }, 1000)  
  }  
}  
delay.print()
```

để fix vấn đề này thì có thể dùng **arrow function**

```
const delay = {  
  lastName: 'Duoc',  
  print() {  
    setTimeout(() => {
```

```
        console.log(this.lastName) // Duoc
      }, 1000)
    }
  }
  delay.print()
```

Lưu ý là **this** trong callback không đề cập đến function chứa callback đó, hãy cẩn thận! **this** dưới đây không đề cập đến **broke** mà nó đề cập đến obj.

```
function broke(func) {
  const obj = {
    name: 'duoc',
    func
  }
  return obj.func()
}

broke(function () {
  console.log(this) // obj
})
```

Vì thế để biết this trong callback đề cập đến cái nào thì phải hiểu được hàm chứa callback gọi callback như thế nào.

Higher order function

Mình đã viết 1 bài chi tiết tại đây: [Chinh phục High Order Function, Closures, Currying và Callback trong Javascript](#)

High order function là một **function** mà nhận vào tham số là **function** hoặc return về một **function**

```
const tinhTong = (a) => (b) => a + b
const ketQua = [1, 2, 3, 4, 5].map((item) => item * item)
console.log(tinhTong(1)(2)) // 3
console.log(ketQua) // [ 1, 4, 9, 16, 25 ]
```

Callback function

Callback function là một **function** mà được truyền vào một **function khác** như một tham số

```
const num = [2, 4, 6, 8]
num.forEach((item, index) => {
  console.log('STT: ', index, 'la ', item)
})
const result = num.map((item, index) => `STT: ${index} la ${item}`)
```

Closure

Closure là cách mà một **function cha** return về một **function con** bên trong nó. Ở trong **function con** đó có thể truy cập và thực thi các biến của **function cha**. Phải đủ 2 điều kiện này mới được gọi là **Closure** nhé.

```
const increase = () => {
  let x = 0
  const increaseInner = () => ++x
  return increaseInner
}
const myFunc = increase()
console.log(increase()) // 1
console.log(increase()) // 1
console.log(myFunc()) // 1
console.log(myFunc()) // 2
console.log(myFunc()) // 3
```

Currying

Currying là một kỹ thuật mà cho phép chuyển đổi một **function nhiều tham số** thành những **function liên tiếp có một tham số**.

```
const findNumber = (num) => (func) => {
  const result = []
  for (let i = 0; i < num; i++) {
    if (func(i)) {
      result.push(i)
    }
  }
  return result
}
findNumber(10)((number) => number % 2 === 1)
findNumber(20)((number) => number % 2 === 0)
findNumber(30)((number) => number % 3 === 2)
```